DOCUMENT RESUME

ED 367 700 TM 021 166

AUTHOR Gratch, Jonathan; And Others
TITLE Rational Learning: Finding A Balance between Utility and Efficiency.
INSTITUTION Illinois Univ., Urbana. Dept. of Computer Science.
SPONS AGENCY National Science Foundation, Washington, D.C.
REPORT NO UILU-ENG-92-1736; UIUCDCS-R-92-1756
PUB DATE Jun 92
CONTRACT NSF-IRI-87-19766
NOTE 20p.
PUB TYPE Reports - Descriptive (141)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS Computer Assisted Instruction; *Computer Uses in Education; Costs; *Efficiency; Learning Strategies; Technological Advancement
IDENTIFIERS *COMPOSER System; Machine Learning; *Rational Learning; Utility Analysis

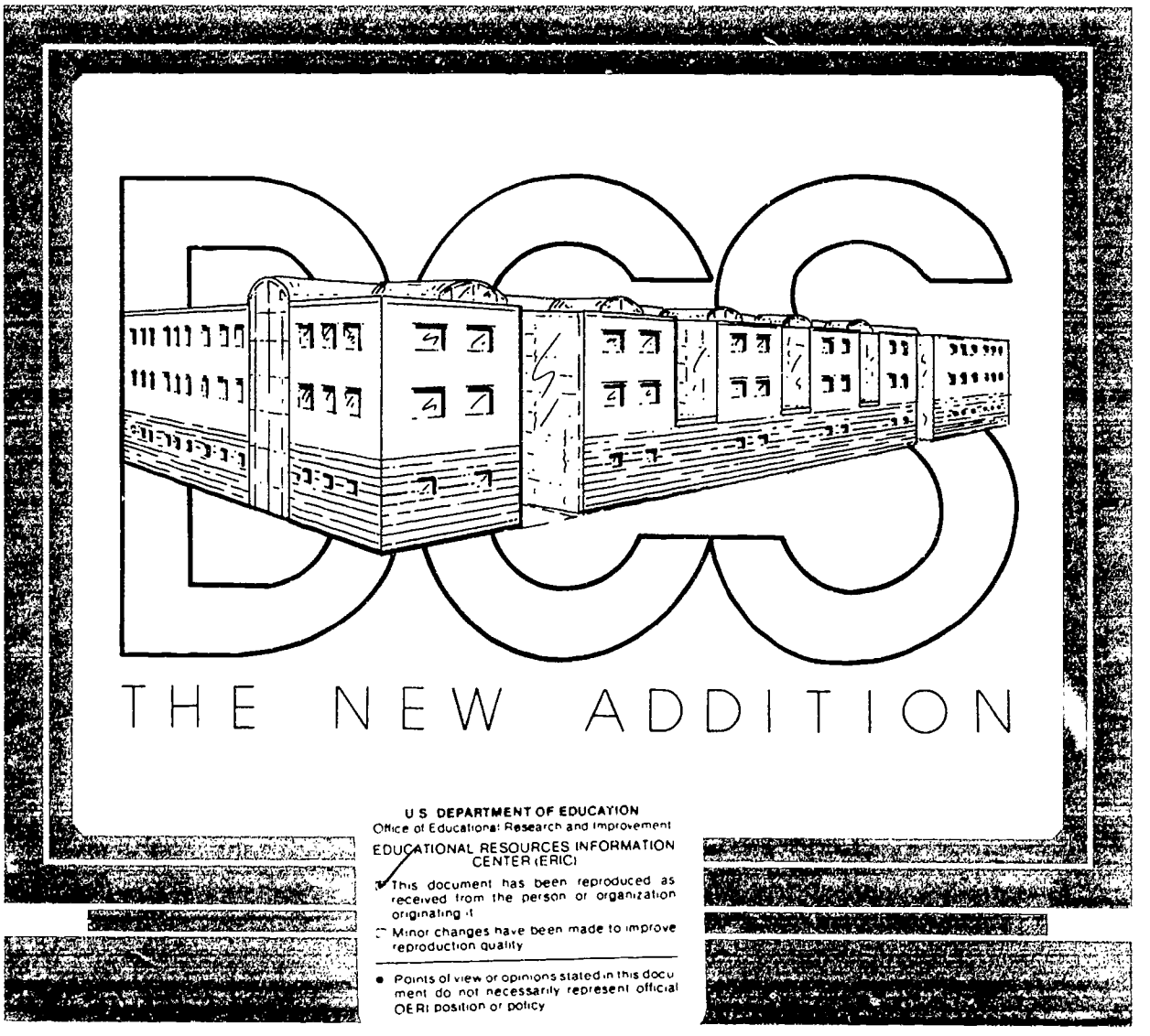
ABSTRACT

The field of machine learning has developed a wide array of techniques for improving the effectiveness of performance elements. Ideally, a learning system would adapt its commitments to the demands of a particular learning situation, rather than relying on fixed commitments that impose tradeoffs between the efficiency and utility of a learning technique. This article presents an extension of the COMPOSER learning approach that dynamically adjusts its learning behavior based on the resources available for learning. COMPOSER is a speed-up learning technique that provides a statistical approach to the utility problem. The system identifies a sequence of transformations that, with high probability, increase the Type I utility of an initial planning system. The approach breaks the task into a learning phase and a utilization phase. This extension to COMPOSER adopts a rational policy that dynamically balances the trade-off between efficiency and utility. Implications for learning systems are discussed. (Contains 24 references.) (SLD)

TM

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN



REPORT NO. UIUCDCS-R-92-1756 UILU-ENG-92-1736

Rational Learning: Finding a Balance Between Utility and Efficiency

by

Jonathan Gratch
Gerald DeJong
Youhong Yang

June 1992

Rational Learning: Finding a Balance Between Utility and Efficiency

Jonathan Gratch and Gerald DeJong
Beckman Institute for Advanced Studies,University of Illinois
405 N. Mathews, Urbana, IL 61801
e-mail: gratch.cs.uiuc.edu

Youhong Yang
Department of Statistics, University of Illinois
725 S. Wright, Urbana, IL 61801

1. INTRODUCTION

The field of machine learning has developed a wide array of techniques for improving the effectiveness of performance elements. Learning techniques are able to take general performance systems and tailor them to the eccentricities of particular domains. In this fashion, slow general systems can be automatically adapted into efficient problem solvers for particular domains. Unfortunately, the task of learning is difficult. Learning systems must operate under limited resources and must make many compromises in the interest of learning efficiency. These compromises appear in the form of design commitments implicit in the architecture of learning systems. These ramifications of these commitments is that they impose tradeoffs between the efficiency and usefulness of a learning technique. The fixed nature of these commitments limits the generality of learning techniques. Ideally, a learning system would adapt its commitments to the demands of a particular learning situation. In this article we present an extension of the COMPOSER learning approach [Gratch92b] which dynamically adjusts its learning behavior based on the resources available for learning.

2. UTILITY-BASED VIEW OF LEARNING

Viewed abstractly, a learning system tailors a performance element to be effective in some environment. We will take the view that a performance element is some procedure which accepts and executes a series of tasks. For example the performance element might be a classifier in which case each input is some feature vector and the output is a classification. Alternatively, the performance element could be a planner where the input is problem specifications; the output is plans. The environment is simply the tasks a performance element faces. Adaptation to this environment must be judged against some criteria for success. For example, in the case of a classifier, success is typically judged in terms of classification accuracy. In planning, criteria include accuracy, planning efficiency, and plan quality.

In this paper we will advocate the utility-based view of learning adopted by several authors [Doyle90, Gratch92b, Greiner92b, Leckie91, Subramanian92]. A particular environment can be characterized by a probability distribution over the set of possible tasks. The user provides a utility function which specifies his criteria for success on individual tasks. The effectiveness of a performance element is characterized by its *expected utility* over the task distribution. This is the sum of the utility of each task weighted by the probability of a task's occurrence. For example, in classification, the standard utility function assigns one to a correctly classified feature vector, and a zero to

an incorrectly classified vector. The expected utility is simply the accuracy of the performance element over the distribution.

Learning can be viewed as a transformational process where, through experience with the environment, some initial performance element, PE_0, is transformed into a performance element, PE_*, with higher expected utility [Gratch92b, Greiner92b]. A classifier can be transformed by updating its representation of the concepts to be classified — for instance through specializing or generalizing transformations. A planner may be transformed by the addition of control knowledge including macro–operators [Braverman88, Laird86, Markovitch89], control rules [Etzioni90, Minton88, Mitchell83], and static board evaluation functions [Utgoff91].

The transformations available to a learner define its *vocabulary of transformations*. These are essentially learning operators and collectively they define a *transformation space*. For instance, acquiring a macro–operator can be viewed as transforming the initial system (the original planner) into a new system (the planner operating with the macro–operator). In [Drummond90], the addition of a reactive rule transforms one subset of the universal plan into another. In [Minton88], a planner's search control strategy is transformed by the addition or deletion of a control rule. A learning technique must explore this space for a sequence of transformations which results in a better planner.

3. LEARNING COST

The utility–based view of learning facilitates a close analogy between learning and work in rational reasoning. In reasoning there is a reasoner which must choose from a set of actions, an action with high expected utility. In the utility–based view of learning, there is a learner which must from choose amongst a set of possibly transformed performance elements, a performance element with high expected utility. In reasoning, a reasoner which always chooses the action with maximal expected utility is *substantively rational* [Simon76] (also called Type 1 rationality [Good71]). Similarly, we can define a *substantively rational learning system* as a system which always identifies the transformed performance element with maximum expected utility.

The substantive rationality is seldom attainable in that it assumes infinite resources. This has led to a focus on rationality under limited resources. Simon refers to this as *procedural rationality* (also called Type 2 rationality [Good71]) because the focus is on identifying efficient procedures for making good–enough decisions. A procedurally rational agent relaxes the strict requirements or substantive rationality in the interest of reasoning efficiency. The analogy between reasoning and learning applies here as well. It is seldom reasonable to expend the resources necessary for a learning system to find optimal solutions. Instead we demand that our learning techniques identify good–enough solutions quickly. Learning techniques embody numerous constraints to achieve tractable behavior. For example, classification learning techniques embody biases to restrict the space of potential transformations. Learning to plan techniques also embody numerous constraints (see [Gratch92a]).

The demands of learning under limited resources can be addressed by two type of deviations from the substantive ideal. In one approach, the generality of a technique can be restricted to special cases. A learning technique retains substantive rationality as long as the learning problem falls within the restricted set of cases. For instance, classification learning techniques can exactly identify the target concept with polynomial examples when it is drawn from a restricted class like monomials or *k–DNF* [Pitt]. But this guarantee only applies if we know in advance that the target concept is a member of one of these restricted classes. If the concept lies outside the class, the techniques can identify sub–maximal representations of concepts.

The second deviation is abandon the goal of maximizing expected utility and instead search for satisfactory, rather than optimal choices. This corresponds to the notion of *satisficing search* [Simon75]. Thus, a learning system can trade–off potential gains from learning in the interest of maintaining efficiency. A strong learning bias can prevent a system from entertaining some of the best transformed performance elements but, hopefully, it can efficiently identify an adequate performance element from the reduced set.

The disadvantage of abandoning substantive rationality is that it afforded a definition of desirable behavior that was independent of the particular procedure which implements the rational behavior. The set of optimal solutions are uniquely defined by the set of choices and the utility function. We lose this uniqueness when we move to procedural rationality. To discuss procedurally rational learning systems we must discuss particular policies for resolving the trade–off between utility and learning cost.

3.1 Fixed Policy

For non–trivial learning problems there is a clear tradeoff between the expected utility of learned performance elements and the efficiency of learning. The typical approach is to adopt a fixed policy towards resolving this tradeoff. The learning system implementor commits to some fixed set of constraints for his or her learning approach. Frequently these constraints are unarticulated and appear implicitly through the learning system architecture. For example, SOAR [Laird86] transforms its planner by acquiring macro–operators or "chunks." A particular domain theory defines the space of possible chunks, but only a subset of possible chunks are actively considered. These are the chunks which arise from problem solving impasses. Furthermore, once a chunk is learned, it can never be forgotten. Thus SOAR is employing a restricted irrecoverable search through the sets of possible chunks. This is clearly more efficient that choosing among all possible sets of chunks, but it is less clear how this policy impacts the potential utility of the resulting planners.

Fixed policies can be quite effective in restricted situations. Unfortunately, the same policy may not apply equally well in all circumstances. We may demand different behavior from our learning system depending on if we have a little or a lot of resources to commit towards learning. The former requires a highly restricted learning technique while latter would be better served by a more liberal policy.

3.2 Parameterized Policy

An alternative to a fixed policy is to allow the user some control over the behavior of the learning technique. This can be seen as incorporating some *degrees of freedom* into the learning technique which must be resolved by the user. An example is the user specified confidence parameter provided by PAC–learning techniques. Higher confidence requires more examples and thus higher learning cost. The user is free to resolve this tradeoff based on the demands of his or her particular circumstances.

Another example, derived by analogy to work in reasoning, is to construct "anytime" learning systems. Anytime algorithms can be interrupted at any point with a useful result [Dean88]. Furthermore, results become monotonically better over time. An anytime learning algorithm must at all times maintain a representation of some viable performance element. As learning resources are expended, the currently represented performance element must monotonically improve in utility. This allows the user to arbitrarily determine the resources to commit to learning.

Parameterized policies can greatly enhance the flexibility of a learning technique, but they also place greater demands on the user. Also, it is not sufficient to provide degrees of freedom. If a learning

technique is to be useful, the user must be told how this freedom impacts the tradeoff between utility and efficiency. Sometimes this information is only available *after* learning has begun. For example, given an anytime learning algorithm, the decision of when to terminate might depend on how fast the current algorithm is improving, or worse, on how fast it will improve if we continue learning. If this information is not available to the user, the additional flexibility is only a burden. Under these circumstances, it is reasonable to build into the learning system capabilities to estimate future learning benefit and provide this information to the user. Thus, adding *useful* flexibility can greatly complicate the task of designing a learning system.

3.3 Rational Policy

Incorporating degrees of freedom to a learning system increases the flexibility of an approach, but it also increases the demands on the user. There may be a quite complex mapping between the goals of the user and the setting of the various learning parameters. Ideally, the user should be able to articulate his or her goals and leave it to the learning system to configure the policy to best satisfy the goals. If the learner can estimate the cost of learning and a expected improvement which results from it, it can use these quantities to dynamically tailor a policy which is suited to the particular learning task. We say a learning system incorporates a *rational policy* if dynamically balances the trade–off between learning utility and learning efficiency. A *rational learner* is a learning system which uses a rational policy.

A rational policy requires the user to explicitly specify the relationship between utility and learning cost. Just as the user of a substantively rational learning approach supplies a utility function (henceforth called a Type 1 utility function) which indicates his or her goals, the user of a procedurally rational learning system must supply a utility function (henceforth called a Type 2 utility function) which indicates how these goals are discounted by the cost to achieve them.

A Type 2 utility function can be surprisingly straightforward. For example, in speed–up learning the problem is to increase the efficiency of a problem solver. Under realistic situations, there are limited resources which must be divided between learning and problem solving. The obvious Type 2 utility function is the expected number of problems which can be solved within a given resource limit. There is some number of problems we can expect to solve with the initial problem solver. If the benefits of learning greatly outweigh the resource cost, it is worthwhile expending some resources towards learning a better problem solver. By maximizing the Type 2 utility function a rational learning system identifies the best tradeoff between learning utility and cost.

4. A SPECIFIC RATIONAL LEARNING TASK

We further explore the issues of rational learning by providing a rational extension of an existing machine learning technique. For this we choose the COMPOSER system [Gratch92b]. COMPOSER is a speed–up learning technique which provides a statistical approach to the utility problem. The system identifies a sequence of transformations which, with high probability, increase the Type 1 utility of an initial planning system. The approach breaks the task into two phase, a learning phase and a utilization phase. First there is a learning phase where examples are taken and transformations adopted. At some point, determined by the user, learning terminates and the user is expected ut utilize the final planner. The learning phase is broken down into a series of stages where after each stage some transformation is adopted.

COMPOSER implements a fixed policy. The space of transformations is explored by greedy hill–climbing. Each new best guess is the result of applying a single transformation to the last guess. Which transformation to adopt is determined by drawing example problems from a fixed problem

distribution, and measuring the change in Type 1 utility afforded by a set of possible transformations. The technique chooses the first transformation which reaches statistical significance using a particular statistical technique. The first transformation to be identified may not be the transformation which provides the greatest change in Type 1 utility. Thus COMPOSER does not employ steepest ascent hill–climbing. On the other hand it requires fewer examples than what would be needed to identify the steepest ascent. This reflects a particular policy on the trade–off between utility and efficiency. COMPOSER stops learning when it exhausts a set of training examples which are provided by the user.

We have recently developed an extension to COMPOSER which does employ steepest ascent. This approach takes sufficient examples at each iteration to identify the transformation which generates a greater increase in Type 1 utility than any other transformation. This can take significantly more examples, and thus significantly more resources, than the original COMPOSER system. Depending on the users goals and available resources, this may or may not be an advance. We could provide a parameter which configures the system to behave somewhere between these two extremes. Unfortunately, the efficiency of the learning system depends on the particular learning task to which it is applied, and this efficiency is generally unknown before the system begins to learn. Thus the user may not be able to make an informed decision on how to set the parameter. For this reason we propose a rational extension of COMPOSER where the degree of freedom is what transformation to adopt at each step, among a choice of alternatives which, with high probability, improve the performance of the current planner. This extension will also internalize the decision of when to stop learning.

Rational learning requires a Type 2 utility function. COMPOSER is a speed–up learning technique which improves the efficiency, but not the accuracy of a planner. We will consider a particular Type 2 function. The learning system should try to maximize the expected number of problems which can be solved after learning, given a fixed set of resources. We call this Type 2 function *ENP* for Expected Number of Problems. One might consider other utility functions, but this one seems reasonable for the class of tasks COMPOSER is intended, and it helps to illustrates several interesting issues that face a rational learner.

We describe a hill–climbing approach to the problem of maximizing *ENP*. The learning algorithm proceeds by a series of stages. In each stage some number of example problems is taken and a decision is made to terminate the learning process or to adopt a transformation. A transformation is adopted if two conditions are satisfied:

i) it enhances the effectiveness of the current performance element with high probability. The acceptable error on the *i*th stage is specified by δ_i.[1]

ii) it produces the greatest expected single–step increase in the expected number of problems which can be solved after learning.

The later is the degree of freedom which the learning system can rationally control. The learning process hill–climbs through a sequence of performance elements, PE_0, PE_1, ..., where each step is expected to be the largest increase in the expected number of problems which can be solved after learning, but there is no guarantee of global optimality. Before each stage the learning system must decide if it should continue to learn. If so, it must decide which transformation best satisfies the above criteria. Section 4.3 describes the implementation, but first we must introduce some notation.

1. The constant $0 < \delta_i < 1$ is the probability that the heuristic added on the *i*th step will improve the *i*th performance element. This can be set such that the total error across all stages is less than some pre–specified constant. See [Greiner92a] for one strategy.

4.1 Sequential Analysis

The problem of identifying beneficial transformations is treated as a problem of statistical inference. The learning system entertains a set of possible transformations. Example problems are drawn randomly according to the fixed problem distribution and statistics are extracted from each example. Many statistical inference procedures are based on a fixed sample size — the number of examples necessary to make a conclusion is determined in advance of any observations. The heart of our technique utilizes a *sequential* statistical procedure [Govindarajulu81]. Sequential procedures differ from fixed–sized techniques in that the sample size is a function of the observations. Sequential procedures provide a test called a *stopping rule* which determines when sufficient examples have been taken. Examples are taken until the stopping rule is satisfied. The number of examples taken when the stopping rule is satisfied is called the *stopping time*. An important advantage of sequential procedures is that the average number of examples required to perform inference is typically smaller the the number required by a fixed–sized technique. This is because a sequential procedure is able to take advantage of the information in the observations to determine the sample size.

First we review standard statistical notation. Let X be a random variable. An observation of a random variable can yield one of a set of possible numeric outcomes where the likelihood of each outcome is determined by an associated probability distribution. X_i is the ith observation of X. EX denotes the expected value of X, also called the mean, μ_X, of the distribution. $\overline{X}_n$ is the *sample mean* and refers to the average of n observations of X. More precisely $\overline{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$. $\overline{X}_n$ is a good estimator for EX.[2]

A measure of the dispersion or spread of a distribution is called the *variance* of a distribution. Variance, denoted σ^2, is defined as the expected squared difference between a single observed outcome for X and the mean of the distribution. Formally, $\sigma^2 = E[(X - \mu_X)^2]$. The *sample variance*, $S_n^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \overline{X}_n)^2$, is a good estimator for the variance of a distribution.

The function $\Phi(x) = \int_{-\infty}^{x} \left(1/\sqrt{2\pi}\right) exp[-0.5y^2]dy$ is the cumulative distribution function of the standard normal (also called standard gaussian) distribution. $\Phi(x)$ is the probability that a point drawn randomly from a standard normal distribution will be less than or equal to x. This function plays a important role in statistical estimation and inference. The Central Limit Theorem shows that, whatever the distribution of X, the function $\sqrt{n}(\overline{X}_n - \mu_X)/\sigma$ [3], approximates the distribution of the standard normal variable (see [Hogg78 pp. 192–195] This approximation is quite good in practice, even with small n. Thus, even when the distribution of X is unknown, in practice we can perform accurate statistical inference using a "normal approximation."

One of the requirements in our problem specification is that each transformation improve, with high probability, the expected utility of the performance element. To satisfy this requirement we rely on a sequential stopping rule. This rule, introduced by Nádas [Nadas69], determines if the mean of a

2. $\overline{X}_n$ is *unbiased* meaning the expected value of $\overline{X}_n$ equals the mean of the distribution. Of all other possible unbiased estimators of the mean, $\overline{X}_n$ has the least variance.

3. If σ^2 is unknown, we can use S^2 instead. This is better approximated with the t–distribution which converges to standard normal as n grows.

distribution is positive or negative where the error of this inference is at most δ. The rule defines a stopping time, *ST*, as:

$$ST = \min_{n \geq n_0} \left(n : \frac{S_n^2}{\bar{X}_n^2} \leq n \frac{1}{a^2} \right)$$

where a is defined by $\Phi(a) = \delta/2$, and n_0 is some predefined positive integer. After stopping, the inference is made that the mean is greater than (less than) zero if the sample mean is greater than (less than) zero. Thus, the sequential procedure first takes a small fixed–sized sample of n_0 examples (typically 3 or 7), and then continues taking examples until the stopping rule is satisfied.

In our approach we evaluate multiple transformations simultaneously. In particular we assign a stopping rule to each of n transformations and let them "race." The winner of the race is transformation with the smallest stopping time. We then base an inference on the results of the winning stopping rule. If the error for each stopping rule is δ, the error of a n–way race is higher. In the worst case the error is $n\delta$.

In our rational extension we must estimate the cost of learning. Given that we are using the Nádas stopping rule, the cost of learning will be a function of the stopping times associated with different transformations (this will be stated more precisely in Section *X*). Thus, one element of an estimate for learning cost is an estimate for stopping times. We can develop an estimator for *ST* using a sample of m examples where $m < ST$ by using S_m^2 and $\bar{X}_m$ as estimators for S_n^2 and $\bar{X}_n$, and solving the inequality within the stopping rule for n. Thus $n = a^2 \frac{S_m^2}{\bar{X}_m^2}$. n must obey the further constraint that it is an integer greater than or equal to n_0. So an estimator for *ST* is:

$$\hat{ST}_m = \max \left\{ n_0, \left\lceil a^2 \frac{S_m^2}{\bar{X}_m^2} \right\rceil \right\}$$

4.2 Implementation Specific Definitions

The algorithm proceeds through a series of stages. Between each stage the algorithm decides if learning should continue. If the decision is to continue, the algorithm must identify a transformation which enhances the number of problems which can be solved after it is acquired. We use the index variable i, $i=0,1,\ldots$ to indicate a specific stage. PE_i denotes the performance element which exists at the start of the ith stage. The user supplies an initial performance element PE_0.

4.2.1 Transformations

At each stage the learning system has some set of transformations it can potentially apply to the current performance element. In the general case, transformations can be added and removed from this set at any point in the learning process. Let $\mathbf{T}_j$ denote the set of transformations available at the start of the jth problem within stage i, and let $\vec{\mathbf{T}}_i$ be a vector which describes how the set of transformations changes within stage i. Typically, $\vec{\mathbf{T}}_i$ depends on the example problems, and thus may not be knowable until after learning. *APPLY* is a function which transforms a performance element with a particular transformation. If a transformation, t, is adopted on the ith stage, the learning system creates a new performance element by applying the transformation; $PE_{i+1} = APPLY(t, PE_i)$.

4.2.2 *Resources*

Each learning stage consumes resources. $\mathbf{R_i}$ denotes the resources remaining at the start of the ith stage. The user supplies an initial resource limit $\mathbf{R_0}$. After each stage the resources available for the subsequent stage are reduced by whatever resources were used.

Let $r_j(PE_i)$ denote a random variable which corresponds to the resources required to solve the jth problem using PE_i. We use the abbreviation r_j where the stage number is unambiguous. $\overline{r_n}(PE_i)$ is the average resource use of PE_i over n problems. This is a good estimator for the mean resource use of PE_i ($E[r(PE_i)]$).

Let $\Delta r_j(t_k|PE_i)$ denote a random variable which corresponds to the incremental utility of transformation t_k on problem j over PE_i. This is the change in resource use that would result on problem j if t_k were applied to PE_i. We use the abbreviation $\Delta r_j(t)$ where the stage number is unambiguous. See [Gratch92b] for one description of how to obtain such values. $\overline{\Delta r_n}(t)$ is the average change in resource use provided by transformation t over n problems. This is a good estimator for the incremental utility of the transformation ($E[\Delta r(t_k|PE_i)]$). We can estimate the expected resource use of the performance element $APPLY(t, PE_i)$ by adding the average resource use of PE_i and the change in resource use of transformation t given PE_i. Formally, $E[r(APPLY(t, PE_i))] \cong \overline{r_n}(PE_i) + \overline{\Delta r_n}(t)$.

4.2.3 *Learning Cost*

A transformation provides some increment of benefit to the expected utility of a performance element. To realize this benefit we must allocate some of the available resources towards learning. Under our statistical formalization of the problem, learning cost is a function of the number of example problems required to learn a transformation, and the cost of processing each example problem. The number of examples depends on our criteria for adoption. For the current task, transformations must improve expected utility with high probability. Using the Nádas stopping rule, the number of examples required is simply the stopping time associated with the transformation, $ST(t)$.

In the general case, the cost of processing the jth problem depends on several factors. It can depend on the particulars of the problem. In can also depend on the currently transformed performance element, PE_i. For example, many learning approaches derive utility statistics by executing (or simulating the execution) of the performance element on each problem. Finally, as potential transformations must be reasoned about, learning cost can depend on the current set of transformations, $\mathbf{T_{ij}}$.

Let $\lambda_j(\mathbf{T_{ij}}, PE_i)$ denote the learning cost associated with the jth problem under the transformation set $\mathbf{T_{ij}}$ and the performance element PE_i. The total learning cost associated with a transformation, t, is the sum of the per problem learning costs over the number of examples needed to apply the transformation. Let $\lambda(t, \vec{\mathbf{T}}_i, PE_i)$ denote the learning cost for transformation t which is defined as $\lambda(t, \vec{\mathbf{T}}_i, PE_i) = \sum_{j=1}^{ST(t)} \lambda_j(\mathbf{T_{ij}}, PE_i)$ where $ST(t)$ is the stopping time associated with transformation t.

4.2.4 *When and What to Learn*

Under our formalization, the task of learning is to maximize the expected number of problems (ENP) which can be solved after learning. Denote this by $ENP(\mathbf{R}, PE)$. This number is a function of the resources which remain after learning and the transformed performance element. Unfortunately we do not know these parameters until learning is complete. We are adopting a hill–climbing approach which simplifies the problem somewhat. At the start of each stage, the learning system only has to

decide if there exists a single transformation which improves the *ENP*. Thus the learning system must estimate the learning cost and benefit of the transformations available on a given stage.

Let $ENP_i(t)$ be the expected number of problems which would result if transformation t was adopted on stage i. Estimating this value is the key to deciding of learning should be performed and if so, which transformation should be applied. We now consider how this can be estimated. If we are in stage i, by definition, $\mathbf{R}_{i+1}$ is the resources available after stage i. PE_{i+1} is the performance element which results from this stage and the expected resource use of PE_{i+1} is $E[r(PE_{i+1})]$. Recall that this is the mean resource cost to solve a problem. The expected number of problems which can be solved with PE_{i+1} given $\mathbf{R}_{i+1}$ resources is simply the ratio of the available resources and the per problem resource use:

$$ENP(\mathbf{R}_{i+1}, PE_{i+1}) = \frac{\mathbf{R}_{i+1}}{E[r(PE_{i+1})]}.$$

PE_{i+1} is the result of transforming PE_i with some transformation t^*. Similarly $\mathbf{R}_{i+1}$ is defined as $\mathbf{R}_i$ minus the the resource cost to learn t^*. The transformation t^* should be the member of $\mathbf{T}_i$ which yields the largest *ENP*. Let us consider the expected number of problems associated with a particular transformation.

Let $ENP_i(t)$ be the expected number of problems which could be solved if, on stage i, t is adopted and learning is immediately terminated. If t were adopted, $PE_{i+1} = APPLY(t, PE_i)$ and the expected resource use of PE_{i+1} is $E[r(APPLY(t, PE_i))] = Er(PE_i) + E\Delta r(t)$. The resources which remain on stage $i+1$ are the resources beginning stage i minus the cost to learn t, or $\mathbf{R}_{i+1} = \mathbf{R}_i - \lambda(t, \vec{\mathbf{T}}_i, PE_i)$. Thus, $ENP_i(t)$ is defied as the ratio of $\mathbf{R}_{i+1}$ to $E[r(PE_{i+1})]$. or:

$$ENP_i(t) = \frac{\mathbf{R}_i - \lambda(t, \vec{\mathbf{T}}_i, PE_i)}{Er(PE_i) - E\Delta r(t|PE_i)}$$

The learning system should pick the transformation which maximizes *ENP*. Thus, t^* is the $t \in \mathbf{T}_i$ such that $ENP_i(t^*) = \max_{t \in \mathbf{T}_i} ENP_i(t)$. Thus, to implement a solution to this rational learning task we must derive an estimator for $ENP(t)$.

4.2.5 *Implementation Specific Assumptions*

Our rational learning approach to this specific learning problem depends on an ability to estimate $ENP_i(t)$. This in turn requires estimators for several parameters which depend on the unknown problem distribution. This includes the resource use of each performance element, $E[r(PE)]$, and the benefit, $E[\Delta r(t|PE)]$, and learning cost, $\lambda(t, \vec{\mathbf{T}}, PE)$, of each potential transformation. This places certain demands on what information must be extracted from each example problem. To estimate resource use and the benefit for each transformation, we require the learning system to determine the resource cost for each problem, $r_j(PE_i)$, and the change in this cost which each transformation could provide, $\Delta r_j(t_k|PE_i)$. The expected resource cost and transformation benefit can be straightforwardly estimated by the sample mean of each of these observations, $\overline{r_n}(PE_i)$ and $\overline{\Delta_n}(t|PE_i)$.

Estimating the learning cost is complicated by the parameter $\vec{\mathbf{T}}_i$ which may not be knowable until after learning is complete. For our first approach to this problem we make a number of simplifying assumptions. We assume that the learning system is provided with a fixed set of transformations. Within a stage, transformations can be discarded from this set, but never added. $\mathbf{T}_i$ indicates the

set which is available at the beginning of each stage. The user or learning system designer supplies an initial set $\mathbf{T}_0$. After each stage i where a transformation is adopted, $\mathbf{T}_{i+1}$ is set to $\mathbf{T}_0$ minus any transformations from $\mathbf{T}_0$ which have already been applied (we assume there is no benefit in applying the same transformation multiple times). This assumption simplifies some of the statistics by ensuring the each transformation has been evaluated over a same sized set of example problems.

We further simplify the problem of estimating learning cost by assuming that the cost to evaluate a problem is independent of which transformations are being evaluated. Let $\lambda_j(PE_i)$ be the cost of processing one example problem and $\lambda(t, PE_i) = \sum_{j=1}^{ST(t)} \lambda_j(PE_i)$ be the cost to learn transformation t. This assumption simplifies the problem of estimating learning cost. Let $\overline{\lambda_n}(PE_i)$ denote the average learning cost across n example problems. Then for any transformation t, if $\hat{ST}_n(t)$ is t's stopping time, $\overline{\lambda_n}(PE_i) \times \hat{ST}_n(t)$ is a good estimator for the cost to learn t. Thus, we can use the following estimator for $\hat{ENP}_i(t)$:

$$\hat{ENP}_i(t) = \frac{\mathbf{R}_i - \overline{\lambda_n}(PE_i) \times \hat{ST}_n(t)}{r_n(PE_i) - \overline{\Delta r_n}(t|PE_i)}$$

We will discuss the consequences of relaxing these assumptions in Section 4.4.

4.3 Implementation

Learning proceeds through a series of stages. Between each stage the system must decide if it is worthwhile to learn for one more stage. If not, control is transferred to the performance element which expends the remaining resources on problem solving. If learning is expected to increase the *ENP*, the learning system must decide what to learn next. As we will see, these two questions are related: deciding whether to learn depends on what is learned.

4.3.1 When to Learn

The system should continue learning for another stage if there exists some transformation of the current performance element which will improve the *ENP*. Let $\mathbf{R}_i$ and PE_i be the current available resources and performance element (initially these are $\mathbf{R}_0$ and PE_0). If learning is terminated at this decision point, the current performance element, PE_i, can solve some expected number of problems with the remaining resources. In particular, we can expect PE_i to solve $\frac{\mathbf{R}_i}{E[r(PE_i)]}$ problems. If the learning system can identify a transformation $t \in \mathbf{T}_i$ with $ENP_i(t)$ greater than this number, learning should proceed for at least one more stage. We will estimate the answer to this question using a fixed-sized statistical inference procedure. The learning system will process a small number of examples and then infer if learning is worthwhile.

For a given stage i, let PE_i randomly select and process n_0 (a predetermined integer) problems. n_0 should be chosen relatively small. Let the learning cost for those problems be $\lambda_1, \lambda_2, \ldots, \lambda_{n_0}$. Let the resource cost for PE_i over the problems be $r_1, r_2, \ldots, r_{n_0}$. Let $\Delta r_1(t_k), \Delta r_2(t_k), \ldots, \Delta r_{n_0}(t_k)$ be the change in resource use over each problem if the transformation t_k is added to PE_i, k = 1, 2, ..., $|\mathbf{T}_i|$.

To determine if learning is worthwhile we will estimate $ENP_i(t)$ for each transformation, based on the n_0 example problems. If we adopt a transformation, it must benefit the performance element with

probability $1 - \delta_i$. Our estimator for $ENP_i(t)$ requires estimators of four values: the mean learning cost, the mean problem solving cost, the mean benefit for the transformation, and the stopping time for the transformation. We base these values on the following statistics, respectively: $\bar{\lambda}_{n_0}$, $\bar{r}_{n_0}$, $\overline{\Delta r}_{n_0}$, and $\hat{ST}_i(t) = max\left\{n_0, \left\lceil \frac{S^2_{n_0}(t)a^2}{[\overline{\Delta r}_{n_0}(t)]^2} \right\rceil \right\}$ where a satisfies $\Phi(a) = \frac{\delta_i}{4|T_i|}$. This selection for a is explained in Section 4.3.2.1.

Learning will help if there exists some $t \in T_i$ which yields an improved ENP. It suffices to consider the transformation with the maximum $\hat{ENP}_i(t)$. Denote this by $\hat{ENP}_1 = \max_{t \in H}\left\{\frac{R_i - \hat{ST}_i(t)\bar{\lambda}_{n_0}}{\bar{r}_{n_0} - \overline{\Delta r}_{n_0}(t)}\right\}$. Recall that this statistic was developed in Section 4.2.5.

$\frac{R_i}{\bar{r}_{n_0}}$ is an estimator for ENP if learning is terminated without adopting a transformation. If this value is larger than $\hat{ENP}_1$, learning should be terminated.

$$\text{Continue learning if: } \hat{ENP}_1 \leq \frac{R_i}{\bar{r}_{n_0}} \qquad \text{(T1)}$$

If this inequality is true, then terminate the process of learning and commence solving problems with PE_i. Intuitively, this means we stop learning if the ENP without learning is higher than the ENP after learning for one more stage. If the test fails (there is some transformation which yields a higher ENP) we learn for at least one more stage.

4.3.2 What to Learn

If the learning system decides that another stage of learning is sanctioned, the system must choose some transformation to adopt. The definition of our learning task imposed two requirements. First, each applied transformation must, with high probability, improve the expected utility of the performance element. Secondly, it must choose a transformation which yields high ENP. We break the decision of what to learn into two steps. First the algorithm attempts to identify a single transformation which improves utility with high probability. If none are discovered, learning is terminated. If such a transformation is discovered, the learning system then decides if it is worthwhile to take an additional set of examples with which to find a transformation with higher $ENP_i(t)$. The error for each of these decisions is set at $\frac{\delta_i}{2}$ so that the total error for the stage is at most δ_i.

4.3.2.1 Phase 1

This phase processes problems, searching for transformations which have positive or negative incremental utility to some pre-specified error level ($\frac{\delta_i}{2}$). Each time a transformation demonstrates negative incremental utility it is removed from T_i. Problems are solved until T_i is exhausted (whereupon learning terminates), or until some transformation demonstrates positive incremental utility. We use the sequential procedure proposed by Nádas [Nadas69].

Take a satisfying $\Phi(a) = \frac{\delta_i}{4|T_i|}$, where δ_i is a predetermined constant $0 < \delta_i < 1$ indicating the acceptable error level for accepting a beneficial transformation in stage i. A particular transformation t

has demonstrated its incremental utility to the specified confidence when $ST(t)$ examples are taken. $ST(t)$ is the stopping time for transformation t and it is determined by the Nádas stopping rule:

$$ST(t) = \max_{n \geq n_0}(n : \frac{S_n^2}{[\overline{\Delta r_n}(t)]^2} \leq \frac{n}{a^2})$$

Let the planner randomly select and solve problems until for at least one transformation, this stopping condition is satisfied. If for at least one such transformation t, the average $\Delta r(t)$ at this point is positive, set t_0 to be the transformation for which the stopping condition is satisfied and for which $\overline{\Delta r}(t)$ is maximum, and proceed to Phase 2. Otherwise, delete all the transformations causing the stopping (these have $\overline{\Delta r}(t) < 0$). Keeping solving problems until either Phase 2 is reached or all the transformations in $\mathbf{T_i}$ are deleted. The later ends the whole process.

From the results in Nádas' paper, for a fixed transformation t, the decision that claiming $E\Delta r(t) > 0$ ($E\Delta r(t) < 0$) if the average when the process stops is positive (negative) has an error probability (approximately) less than or equal to $\frac{\delta_i}{2|\mathbf{T_i}|}$. By Bonferroni's method, the error probability at the end of Phase 1 of claiming $E\Delta r(t) > 0$ while it is negative, or deleting a transformation with positive $E\Delta r(t)$ is (approximately) less than or equal to $\frac{\delta_i}{2}$.

4.3.2.2 Phase 2

Following Phase 1, t_0 is a transformation with positive incremental utility with small error probability ($\frac{\delta_i}{2}$). Other members of $\mathbf{T_i}$ might yield a higher *ENP* but they have yet to demonstrate significance. The purpose of Phase 2 is to decide between adopting t_0 immediately, or to solve an additional set of problems which will allow these other potentially better transformations to reach significance. The test of Dudewicz and Dalal [Dudewicz75] tells us the number of additional problems which have to be solved to determine if another transformation has greater incremental utility t_0 with error probability $\frac{\delta_i}{2}$. This number can be used to determine the *ENP* for these other transformations. If no other transformation has greater *ENP* then t_0 we adopt t_0 immediately and proceed to the next stage. Otherwise we determine some subset of $\mathbf{T_i}$ which is worth investigating further.

The Dudewicz and Dalal procedure is designed to choose, from among a population of K random variables, the random variable with the highest mean. The procedure identifies the correct variable (the one with the highest mean) with probability p^*. whenever the difference between the top two means is greater than or equal to some value ε. In the case where the difference is less than ε, the procedure may not select the best mean, but in this case we, with high probability, select a mean which is "ε–close."

The procedure is based on a multi–variate t–distribution. This plays an role analogous to the standard normal distribution in the Nádas technique. Instead of using the constant a, Dudewicz and Dalal define the constant h as $h = h_m(K, p^*)$ be the unique solution of $\int_{-\infty}^{\infty} [F_m(x+h)]^{K-1} f_m(x)dx = p^*$ where p^* is the probability of a correct decision, K is the number of random variables, and $F_m(\cdot)$ and $f_m(\cdot)$

are the cumulative distribution function and density respectively of a student–t random variable with $m = ST - 1$ degrees of freedom. $f_m(x) = \frac{\Gamma[(m+1)/2]}{\sqrt{\pi m}\,\Gamma(m/2)(1+x^2/m)^{(m+1)/2}}$, $F_m(x) = \int_{-\infty}^{x} f_m(w)dw$, $\Gamma(t) = \int_0^{\infty} y^{t-1}e^{-y}dy$.

In our problem, $K = |\mathbf{T_i}|$ and $p^* = 1 - \frac{\delta}{2}$ where $\frac{\delta_i}{2}$ is the acceptable error for this phase. The table of $h_m(K, p^*)$ is given in [Dudewicz75].

In an analogous fashion to the estimate for the stopping time for the Nádas technique, Dudewicz and Dalal define the number of examples to pick the highest mean with a stopping time. We use this to determine the additional number of examples required for each transformation. Call the number of examples $\hat{ST}_2(t)$. $\hat{ST}_2(t) = max\left\{ST+1, \left\lceil\left(\frac{S_{ST}(t)h_i}{\epsilon}\right)^2\right\rceil\right\}, t \neq t_0$ where ε is a predetermined number. This is the number of problems which are required to test if $t \neq t_0$ has an incremental utility ε greater than t_0. First we will test if any transformation has higher *ENP* than t_0. This is again based on the statistics for $ENP_i(t)$ that we developed in Section NO TAG, but instead of using the expected stopping time from the Nádas rule, we use the number of examples derived from the Dudewicz and Dalal test:

$$\hat{ENP}_2 = \max_{t \neq t_0}\left\{\frac{\mathbf{R}_i - \hat{ST}_2(t)\bar{\lambda}_{ST}}{\bar{r}_{ST} - \overline{\Delta r}_{ST}(t)}\right\}$$

$\hat{ENP}_2$ is the maximum estimate of $ENP_i(t)$ of transformations other than t_0.

$$\text{Adopt } t_0 \text{ if } \hat{ENP}_2 \leq \frac{\mathbf{R}_i - ST\bar{\lambda}_{ST}}{\bar{r}_{ST} - \overline{\Delta r}_{ST}(t_0)} \qquad \text{(T2)}$$

In this case the system places t_0 into the strategy set of the planner which finishes this stage. In this case we do not expect to do better than t_0.

If there is at least one transformation with sufficiently high $\hat{ENP}_2$, we want to find some subset of $\mathbf{T_i}$ which is worthwhile evaluating. The Dudewicz and Dalal technique defines a number of example problems we must take to certify that a transformation has higher incremental utility than t_0, namely $\hat{ST}_2(t)$. If we choose to evaluate some transformation t, we are forced to take at least $\hat{ST}_2(t)$ examples. If we evaluate a set of transformations then we are forced to take a number of examples equal to the maximum $\hat{ST}_2(t)$ of that set. Thus, while we may produce a high *ENP* by evaluating the entirety of $\mathbf{T_i}$, it may be worthwhile to consider some subset of the available transformations.

Let t_1 be the transformation with maximum $\hat{ENP}_2$ ($t_1 = \max_{t \neq t_0} \hat{ENP}_2(t)$). The transformation t_1 will produce the highest expected gain in *ENP*.

$$\text{Let } ST_2 = \max\left\{ST+1, \left\lceil\left(\frac{S_{ST}(t_1)\delta}{\delta^*}\right)^2\right\rceil\right\}$$

This is the number of examples required to decide if t_1 is truly better than t_0. t_1 may not be better than t_0 and other members of $\mathbf{T_i}$ may. Therefore we want to continue evaluating any potentially beneficial transformations which can demonstrate significant improvement within ST_2 problems.

$$\text{Let } T_1 = \{t \in T_1, ST_2(t) \le ST_2\} \bigcup t_0 \qquad \text{(T3)}$$

T_1 contains t_0 plus all transformations for which we can determine a significantly higher incremental utility over t_0 given ST_2 problems. Let $ST_3 = ST_2 - ST$. This is the additional number of problems which must be processed. Let the planner select and process another ST_3 problems. The system then adopts the transformation from T_1 with the largest $\tilde{\Delta r}(t)$. Here, $\tilde{\Delta r}(t)$ is a weighted average of the observations $\Delta r_j(t)$, $1 \le j \le ST_1(t), t \in T_1$. $\tilde{\Delta r}(t)$ is given by

$$\tilde{\Delta r}(t) = \sum_{j=1}^{ST_2} a_j(t)\Delta r_j(t)$$

the $a_j(t)$ being subject to the conditions

$$a_1(h) = \ldots = a_{n_2} \qquad \sum_{j=1}^{ST_3} a_j(t) = 1$$

$$\text{and } S^2_{ST}(t) \sum_{j=1}^{ST_3} a_j^2(t) = \left(\frac{\delta^*}{\delta}\right)^2$$

Dudewicz and Dalal suggest the following strategy for setting $a_j(t)$:

$$a_1(t) = \ldots = a_{ST_3-1} = \beta$$

$$a_{n_3} = 1 - (n_3 - 1)\beta$$

$$\beta \in \left\{\frac{(ST_3 - 1) \pm \sqrt{(ST_3-1)^2 - (ST_3-1)ST_3\left(1 - (\delta^*/\delta)^2/S^2_{ST_3}(t)\right)}}{(ST_3-1)ST_3}\right\}$$

It was proved in Dudewicz, E. J. and Dalal, S. R. (1975) that if the difference between the largest $E\Delta r(t)$ and the second largest $E\Delta r(t)$ in T_1 is bigger than or equal to δ^*, assume also that $\Delta r_j(t)$ are all independent, then the probability of selecting the transformation with the largest $E\Delta r(t)$ is no less than p^*. This is a reasonable test even if $\Delta r_j(t)$ are not independent. Thus, as t_0 has positive incremental utility ($\overline{\Delta r_{n_2}}(t_0) > 0$) with error probability (approximately) at most $\frac{\delta_i}{2}$ and the transformation we adopt has greater has incremental utility (if different than t_0) with error probability (approximately) at most $\frac{\delta_i}{2}$, we adopt a transformation with positive incremental utility with error probability (approximately) at most δ_i. If the several top means are very close to each other (the difference is less than δ^*), we can not assure that the transformation we choose has higher incremental utility than h_o, but intuitively, the procedure is unlikely to select a transformation with $E\Delta r(t)$ far away from the best. In this case, we do not lose much even if we do not get the best transformation.

After adopting this transformation, the learning system re–initializes T_{i+1} to any non–adopted transformations and decides again if learning should proceed.

In summary, learing proceeds through a series of stages. Within each stage the system must make two rational decisions. First the system takes a small set of example and decides determine if further learning would improve Type 2 utility. If so, some number of examples are taken to find a transformation meeting the minimal requirements. Next, the system decides if this transformation should

be adopted immediately or if an aditional phase of learning is likely to improve Type 2. The stage terminates after adompting a transformation, or when it is realized that no transformation is likely to improve Type 2 utility. The learning system iterates through a series of stages until a decision is made to terminate learning.

4.4 Discussion

This procedure proceeds through one or more stages, producing a performance element modified with zero or more transformations. This continues until either the system decides it is not worthwhile to learn, or all transformations are discarded in Phase 1. On each stage for which a transformation is adopted, the new performance element will be more effective with error δ_i. On average the new performance element will also produce a larger *ENP*, but we cannot yet characterize a bound on this probability. We are investigating numerical simulations of the technique and expect results to be available soon.

We adopted a number of assumptions in this presentation. We assumed that the learning cost withing a stage is independent the number of transformations. This may not be realistic as in speed–up learning systems like COMPOSER where the cost of extracting incremental utility values depends on the number of transformations. The model can be extended by breaking learning time into two multiple components — time spent solving each training problem, time spent processing each training problem, and an additional transformation specific cost which is the additional time required to evaluate each transformation. The transformation specific costs complicates the decision of whether to proceed learning, for, although a set of transformations might yield a low maximum *ENP*, some subset of those transformations would have a lower per problem learning cost, and might yield a high *ENP*.

The other major assumption was that the set of trasformations does not grow withing a stage. This also does not realistic as most speed–up learning systems consider new transformations throughout the learning phase. The simplest approach would be to add an initial phase where transformations were learned. A difficulty would be in applying rationality to deciding if it were worth learning a new transformations. This is because we have no information of the expected improvement of a transformation until we learn it. We could avoid this complication by moving to a Bayesian model. The learning system would then need to incorporate prior expectations on the benefit of learning a transformation.

5. CONCLUSION

Learning systems cannot produce maximal increases in performance *and* be maximally efficient. Instead it must adopt a policy which balances these two needs. Most learning techniques adopt a particular policy to this tradeoff. Unfortunately, fixed policies limit the generality of learning techniques. In this article we have tentatively explored the issue of rational learning policies and we described an extension to the COMPOSER system which adopts a rational policy. While this is only a first attempt at the problem, but it raises a number of interesting issues, and points to possible solutions for many of them. More importantly, it highlights an issue which is not sufficiently discussed in the learning community — the trade–off between learning efficiency and utility.

Acknowledgements

We are indebted to the help of Yuhong Yang, Bassirou Chitou, Wen Bin Zhang, and John Marden. This research is supported by the National Science Foundation, grant NSFIRI 87–19766.

References

[Braverman88] M. S. Braverman and S. J. Russell, "IMEX: Overcoming intractability in explanation based learning," *Proceedings of the National Conference on Artificial Intelligence*, St. Paul, MN, 1988, pp. 575–579.

[Dean88] T. Dean and M. Boddy, "An Analysis of Time–Dependent Planning," *Proceedings of The Seventh National Conference on Artificial Intelligence*, Saint Paul, MN, August 1988, pp. 49–54.

[Doyle90] J. Doyle. "Rationality and its Roles in Reasoning (extended version)," *Proceedings of the National Conference on Artificial Intelligence*, Boston, MA, 1990, pp. 1093–1100.

[Drummond90] M. Drummond and J. Bresina, "Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction," *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, August 1990, pp. 138–144.

[Dudewicz75] E. J. Dudewicz and S. R. Dalal, "Allocation of Observations in Ranking and Selection with Unequal Variances," *Sankhya: The Indian Journal of Statistics 37, Series B, Pt. 1*, (1975), pp. 28–78.

[Etzioni90] O. Etzioni, "Why Prodigy/EBL Works," *Proceedings of the National Conference on Artificial Intelligence*, Boston, MA, August 1990, pp. 916–922.

[Good71] I. J. Good, "The Probabilistic Explication of Information, Evidence, Surprise, Causality, Explanation, and Utility," in *Foundations of Statistical Inference*, V. P. Godambe, D. A. Sprott (ed.), Hold, Rienhart and Winston, Toronto, 1971, pp. 108–127.

[Govindarajulu81] Z. Govindarajulu, *The Sequential Statistical Analysis*, American Sciences Press, INC., Columbus, OH, 1981.

[Gratch92a] J. Gratch and G. DeJong, "A Framework of Simplifications in Learning to Plan," *First International Conference on Artificial Intelligence Planning Systems*, College Park, MD, 1992.

[Gratch92b] J. Gratch and G. DeJong, "COMPOSER: A Probabilistic Solution to the Utility Problem in Speed–up Learning," *Proceedings of the National Conference on Artificial Intelligence*, San Jose, CA, July 1992.

[Greiner92a] R. Greiner and W. W. Cohen, "Probabilistic Hill–Climbing," *Proceedings of Computational Learning Theory and 'Natural' Learning Systems*, 1992. ((to appear))

[Greiner92b] R. Greiner and I. Jurisica, "A Statistical Approach to Solving the EBL Utility Problem," *Proceedings of the National Conference on Artificial Intelligence*, San Jose, CA, July 1992.

[Hogg78] R. V. Hogg and A. T. Craig, *Introduction to Mathematical Statistics*, Macmillan Publishing Co., Inc., London, 1978.

[Laird86] J. E. Laird, P. S. Rosenbloom and A. Newell, *Universal Subgoaling and Chunking: The Automatic Generation and Learning of Goal Hierarchies*, Kluwer Academic Publishers, Hingham, MA, 1986.

[Leckie91] C. Leckie and I. Zukerman, "Learning Search Control Rules for Planning: An Inductive Approach," *Proceedings of the Eighth International Workshop on Machine Learning*, Evanston, IL, June 1991, pp. 422–426.

[Markovitch89] S. Markovitch and P. D. Scott, "Utilization Filtering: a method for reducing the inherent harmfulness of deductively learned knowledge," *Proceedings of The Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 738–743.

[Minton88] S. N. Minton, "Learning Effective Search Control Knowledge: An Explanation–Based Approach," Ph.D. Thesis, Department of Computer Science, Carnegie–Mellon University, Pittsburgh, PA, March 1988. (Also appears as CMU–CS–88–133)

[Mitchell83] T. M. Mitchell, P. E. Utgoff and R. Banerji, "Learning by Experimentation: Acquiring and Refining Problem–solving Heuristics," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, T. M. Mitchell (ed.), Tioga Publishing Company, Palo Alto, CA, 1983. pp. 163–190.

[Nadas69] A. Nadas, "An extension of a theorem of Chow and Robbins on sequential confidence intervals for the mean," *The Annals of Mathematical Statistics 40*, 2 (1969), pp. 667–671.

[Pitt] L. Pitt and L. G. Valiant, "Computational Limitations on Learning from Examples," *Journal of the Association for Computing Machinery 35*, 4 pp. 965–984.

[Simon75] H. A. Simon and J. B. Kadane, "Optimal problem–solving search: all–or–none solutions," *Artificial Intelligence 6*, (1975), pp. 235–247.

[Simon76] H. A. Simon, "From Substantive to Procedural Rationality," in *Method and Appraisal in Economics*, S. J. Latsis (ed.), Cambridge University Press, 1976, pp. 129–148.

[Subramanian92] D. Subramanian and S. Hunter, "Measuring Utility and the Design of Provably Good EBL Algorithms," *Proceedings of the Ninth International Conference on Machine Learning*, Aberdeen, Scotland, July 1992.

[Utgoff91] P. E. Utgoff and J. A. Clouse, "Two kinds of training information for evaluation function learning," *Proceedings of the National Conference on Artificial Intelligence*, Anaheim, CA, July 1991, pp. 596–600.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-92-1756	2.	3. Recipient's Accession No.
4. Title and Subtitle Rational Learning: Finding a Balance Between Utility and Efficiency			5. Report Date June 1992 6.
7. Author(s) Jonathan Gratch, Gerald DeJong, Youhong Yang			8. Performing Organization Rept. No. R-92-1756
9. Performing Organization Name and Address Department of Computer Science University of Illinois 1304 W. Springfield Avenue Urbana, IL 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No. NSF IRI 87-19766
12. Sponsoring Organization Name and Address NSF			13. Type of Report & Period Covered Technical 14.

15. Supplementary Notes

16. Abstracts

The field of machine learning has developed a wide array of techniques for improving the effectiveness of performance elements. Learning techniques are able to take general performance systems and tailor them to the eccentricities of particular domains. In this fashion, slow general systems can be automatically adapted into efficient problem solvers for particular domains. Unfortunately, the task of learning is difficult. Learning systems must operate under limited resources and must make many compromises in the interest of learning efficiency. These compromises appear in the form of design commitments implicit in the architecture of learning systems. These ramifications of these commitments is that they impose tradeoffs between the efficiency and usefulness of a learning technique. The fixed nature of these commitments limits the generality of learning techniques. Ideally, a learning system would adapt its commitments to the demands of a particular learning situation. In this article we present an extension of the COMPOSER learning approach [Gratch92b] which dynamically adjusts its learning behavior based on the resources available for learning.

17. Key Words and Document Analysis. 17a. Descriptors

Utility problem
Machine Learning
Statistics
Decision theory

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
unlimited	UNCLASSIFIED	18
	20. Security Class (This Page) UNCLASSIFIED	22. Price

FORM NTIS-35 (10-70)

USCOMM-DC 40329-P71